초급 프로젝트

HR BANK

SB_2기 6팀

목차

01 프로젝트 개요

프로젝트 소개

HR BANK

기업의 인적 자원을
안전하게 관리하는 인사 관리 시스템

프로젝트 선택 이유

1.대용량 데이터 처리
(N+1 문제, cursor pagination 경험)

2. 파일 시스템에 대한 이해

3. 스케쥴러를 이용한 배치처리 경험

4. 캐시/인덱스 최적화 경험

02 개발 환경 및 기술 스택

개발 환경

IDE

협업 도구

Discord

GitHub


notion

일정 관리


GitHub Issues

API 테스트 도구


POSTMAN

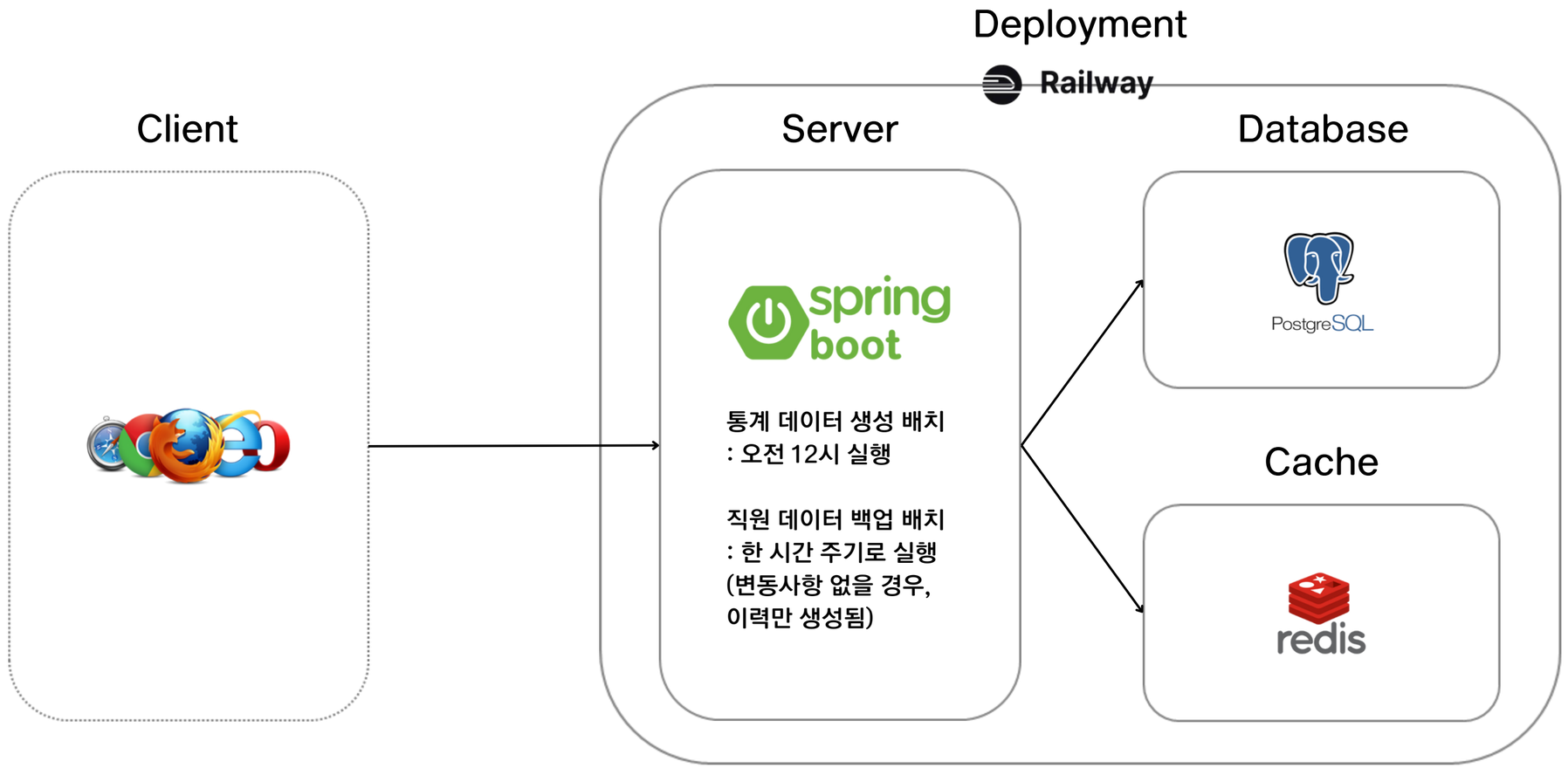
DB 관리 도구


docker

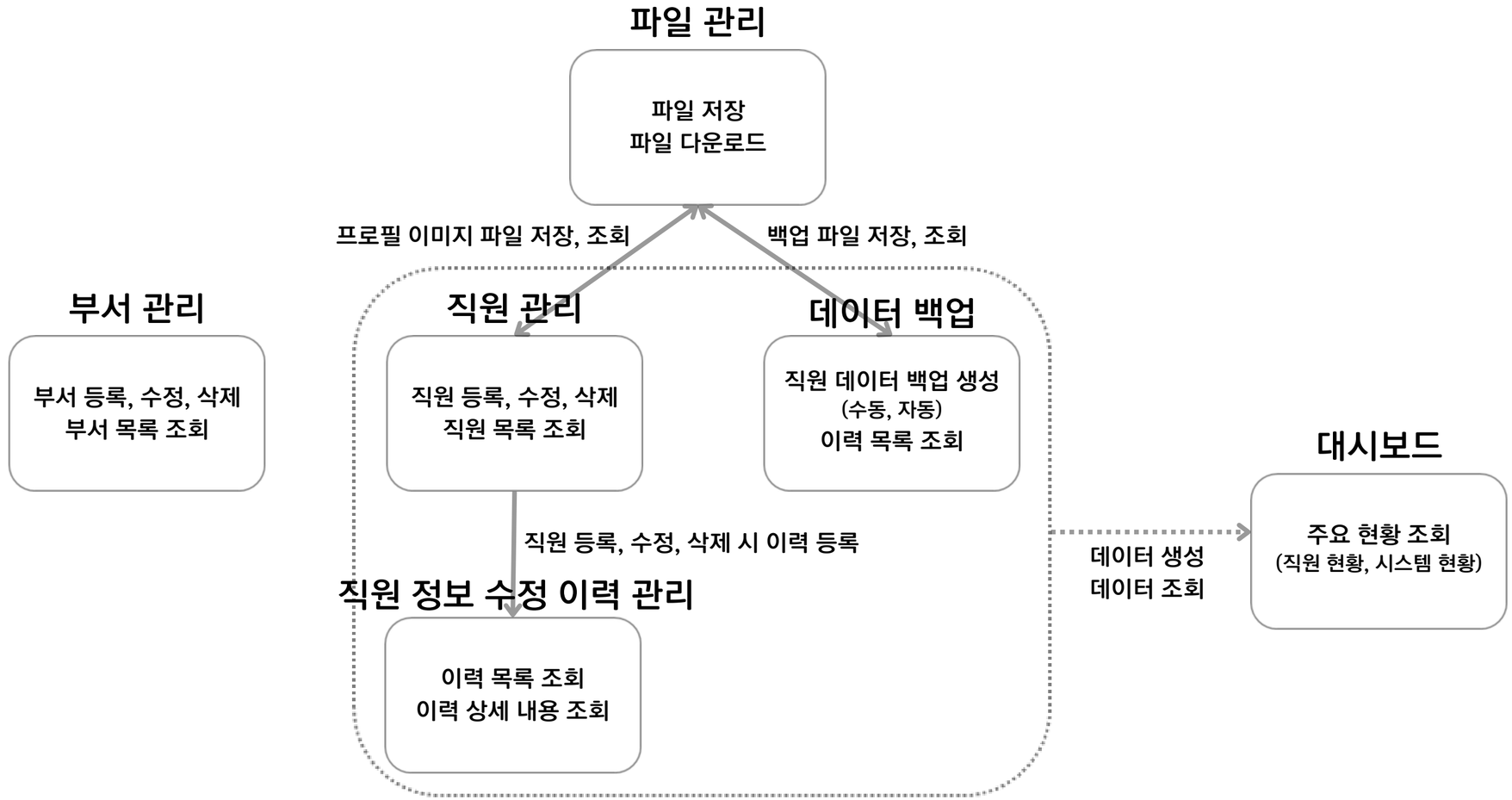
02 개발 환경 및 기술 스택

기술 스택

03 프로젝트 구조


Deployment
Railway
Client
Server
Database
spring boot
통계 데이터 생성 배치
: 오전 12시 실행
직원 데이터 백업 배치
: 한 시간 주기로 실행
(변동사항 없을 경우,
이력만 생성됨)
PostgreSQL
Cache
redis

03 프로젝트 구조


파일 관리
파일 저장
파일 다운로드
프로필 이미지 파일 저장, 조회
백업 파일 저장, 조회
부서 관리
부서 등록, 수정, 삭제
부서 목록 조회
직원 관리
직원 등록, 수정, 삭제
직원 목록 조회
데이터 백업
직원 데이터 백업 생성
(수동, 자동)
이력 목록 조회
직원 등록, 수정, 삭제 시 이력 등록
직원 정보 수정 이력 관리
이력 목록 조회
이력 상세 내용 조회
데이터 생성
데이터 조회
대시보드
주요 현황 조회
(직원 현황, 시스템 현황)

04 팀원 구성 및 협업 방식

R&R

공한나

- 수정 이력 관리 기능
- 공통 모듈
- 팀문서 검토
- 발표 자료 작성 보조

김현호

- 데이터 백업 관리 기능
- 파일 관리 기능
- README 작성

도효림

- 직원 정보 관리 기능
- 발표 자료 작성

양성준

- 직원 통계 조회 기능
- 대시보드
- Railway 배포

최규원

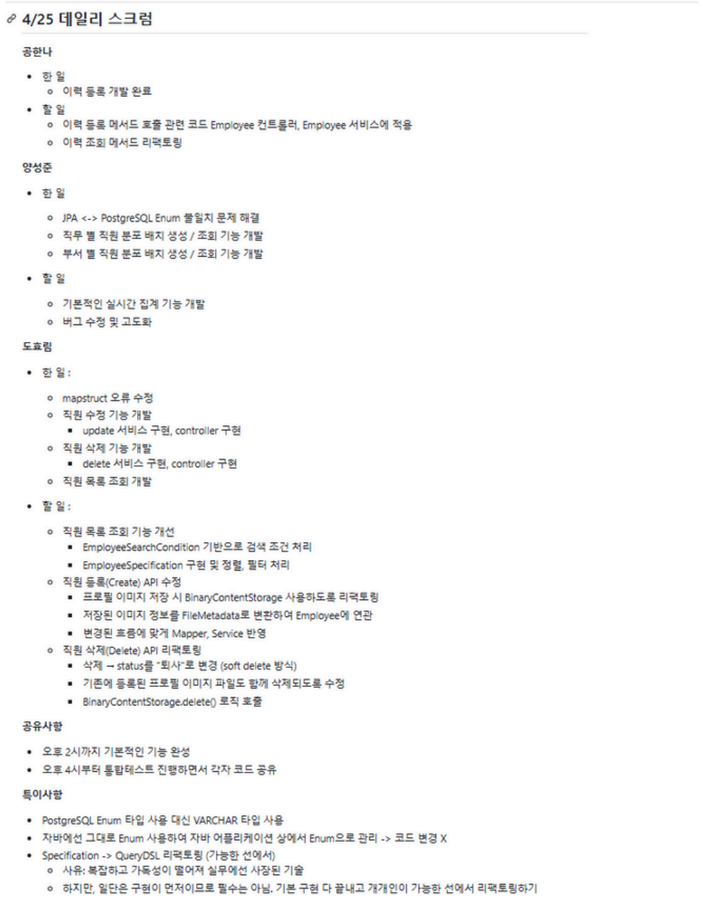
- 부서 정보 관리 기능
- 시연 영상 촬영

04 팀원 구성 및 협업 방식

협업 방식 – 이슈 트래킹 및 일정 관리

데일리 스크럼

- 데일리 스크럼
  - 4/22 데일리 스크럼
  - 4/23 데일리 스크럼
  - 4/24 데일리 스크럼
  - 4/25 데일리 스크럼
  - 4/26 데일리 스크럼
  - 4/28 데일리 스크럼



4/25 데일리 스크럼

공한나

- 한 일
  - 이력 등록 개발 완료
- 할 일
  - 이력 등록 메서드 호출 관련 코드 Employee 컨트롤러, Employee 서비스에 적용
  - 이력 조회 메서드 리팩토링

양성준

- 한 일
  - JPA <-> PostgreSQL Enum 뭘할지 문제 해결
  - 직무 별 직원 분포 배지 생성 / 조회 기능 개발
  - 부서 별 직원 분포 배지 생성 / 조회 기능 개발
- 할 일
  - 기본적인 실시간 집계 기능 개발
  - 버그 수정 및 고도화

도효림

- 한 일 :
  - mapstruct 오류 수정
  - 직원 수정 기능 개발
    - update 서비스 구현, controller 구현
  - 직원 삭제 기능 개발
    - delete 서비스 구현, controller 구현
  - 직원 목록 조회 개발
- 할 일 :
  - 직원 목록 조회 기능 개선
    - EmployeeSearchCondition 기반으로 검색 조건 처리
    - EmployeeSpecification 구현 및 정렬, 필터 처리
  - 직원 등록(Create) API 수정
    - 프로필 이미지 저장 시 BinaryContentStorage 사용하도록 리팩토링
    - 저장된 이미지 정보를 FileMetadata로 변환하여 Employee에 연관
    - 변경된 흐름에 맞게 Mapper, Service 반영
  - 직원 삭제(Delete) API 리팩토링
    - 삭제 → status를 "퇴사"로 변경 (soft delete 방식)
    - 기존에 등록된 프로필 이미지 파일도 함께 삭제되도록 수정
    - BinaryContentStorage.delete() 로직 호출

공유사항

- 오후 2시까지 기본적인 기능 완성
- 오후 4시부터 통합테스트 진행하면서 각자 코드 공유

특이사항

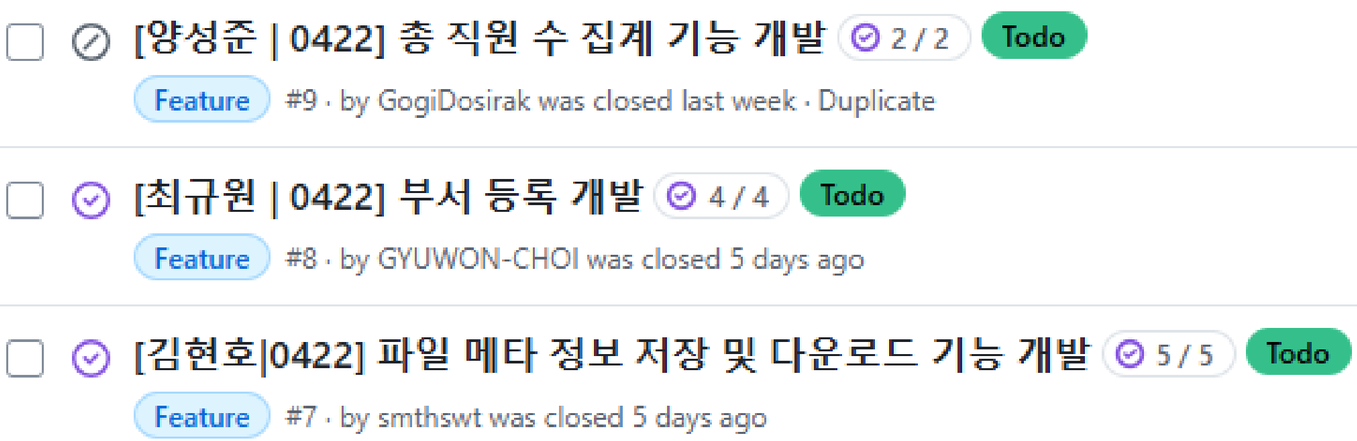
- PostgreSQL Enum 타입 사용 대신 VARCHAR 타입 사용
- 자바에선 그대로 Enum 사용하여 자바 어플리케이션 상에서 Enum으로 관리 -> 코드 변경 X
- Specification -> QueryDSL 리팩토링 (가능한 선에서)
  - 사유: 복잡하고 가독성이 떨어져 실무에선 사장된 기술
  - 하지만, 일단은 구현이 먼저이므로 필수는 아님. 기본 구현 다 끝내고 개개인이 가능한 선에서 리팩토링하기

04 팀원 구성 및 협업 방식

협업 방식 – 이슈 트래킹 및 일정 관리

이슈 관리

‘[이름 | 이슈 생성 날짜] 작업 내용 요약’ 형식으로 관리



04 팀원 구성 및 협업 방식

협업 방식 – 이슈 트래킹 및 일정 관리

서브 이슈 관리

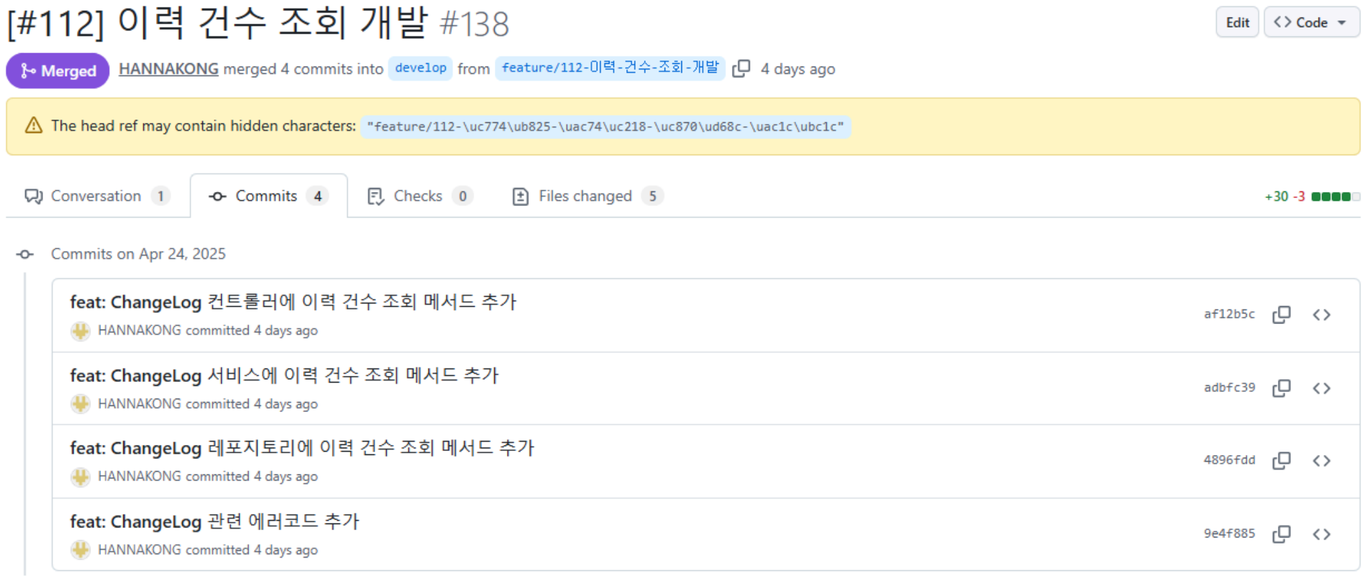
Commit 단위의 작업 내용으로 서브 이슈 관리

04 팀원 구성 및 협업 방식

협업 방식 – 이슈 트래킹 및 일정 관리

커밋 단위

기능 및 메서드 단위로 커밋



04 팀원 구성 및 협업 방식

협업 방식 – 이슈 트래킹 및 일정 관리

PR 단위 기본 기능 구현/리팩토링/버그 수정 작업 단위로 PR 관리

- [ ] [#211] 간단한 코드 리팩토링 및 로깅 추가
  #214 by GogiDosirak was merged 9 hours ago
- [ ] [#199] 직원 등록/조회 기능 개선 및 페이지네이션 오류 수정
  #210 by coderimspace was merged 9 hours ago
- [ ] [#204] 페이지네이션 구현 보완, N+1 문제 해결 refactoring
  #209 by HANNAKONG was merged 9 hours ago
- [ ] [#193] 부서 한글 정렬 오류 수정 및 유효성 검증 추가
  #198 by GYUWON-CHOI was merged 9 hours ago

04 팀원 구성 및 협업 방식

협업 방식 – 이슈 트래킹 및 일정 관리

브랜치 형식

‘작업 유형/이슈번호-작업내용’ 형식으로 브랜치 관리

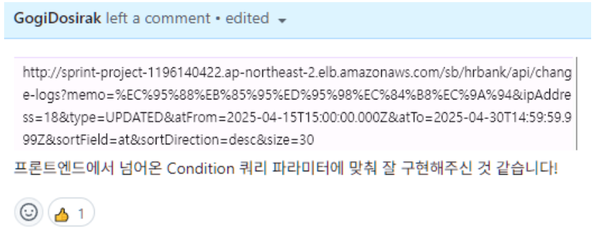
`develop` from `feature/66-직원-상세조회`

`develop` from `feature/179-부서-목록조회`

`develop` from `feature/221-API-문서화`

04 팀원 구성 및 협업 방식

협업 방식 – 코드 리뷰 방식



PR 생성 시, 전원 코드 리뷰 후 approve 처리해야 merge 가능

협업 방식 – 개발 컨벤션

네이밍 컨벤션

-변수, 함수: camelCase
-클래스: PascalCase
-파일: kebab-case
-패키지명: com.team6.hrbank(소문자)
-테이블명: snake_case(소문자, 복수형)
-컬럼명: snake_case(소문자)

커밋 컨벤션

-feat: 새로운 기능 추가
-fix: 버그 수정
-refactor: 코드 리팩토링
-chore: 기타 변경사항

브랜치 전략

변형된 Git Flow 전략 적용:
-main (배포)
-develop (통합)
-feature/기능명 (기능 개발)
-bugfix/기능명 (버그 수정)

예외 처리

-Enum 파일로 예외 코드 관리
+ GlobalExceptionHandler로
전역 예외 처리

05 프로젝트 수행 절차

1차	2차	3차	4차
프로젝트 기획	1차 개발 스프린트	중간 점검	2차 개발 스프린트
프로젝트 계획 요구사항 정리 ERD & 엔티티 설계 프로젝트 세팅	주요 기능 구현	통합 테스트 수행	피드백 반영 및 고도화

05 프로젝트 수행 방법

API 명세

Employee-Controller 직원 관련 API

GET	/api/employees	직원 목록 검색
POST	/api/employees	직원 생성
GET	/api/employees/{id}	직원 단건 조회
DELETE	/api/employees/{id}	직원 삭제
PATCH	/api/employees/{id}	직원 정보 수정
GET	/api/employees/count	직원 수 조회

ChangeLog-Controller 직원 정보 변경 이력 관리 API

GET	/api/change-logs	직원 정보 변경 이력 목록 조회
GET	/api/change-logs/{changeLogId}/diffs	변경 이력의 상세 내용 조회
GET	/api/change-logs/count	변경 이력 건수 조회

DashBoard-Controller 대시보드 관련 API

GET	/api/employees/stats/trend	직원 수 전체 추이 조회
GET	/api/employees/stats/distribution	직무/부서 별 직원 분포 조회

https://sb02-hrbank-team6-production.up.railway.app/swagger-ui/index.html

05 프로젝트 수행 방법

API 명세

Department-Controller 부서 관련 API

GET	/api/departments	부서 목록 조회
POST	/api/departments	부서 생성
GET	/api/departments/{id}	부서 조회
DELETE	/api/departments/{id}	부서 삭제
PATCH	/api/departments/{id}	부서 수정

Backup-Controller 데이터 백업 관련 API

GET	/api/backups	데이터 백업 목록 조회
POST	/api/backups	데이터 백업 생성
GET	/api/backups/latest	최근 백업 이력 조회

FileMetadata-Controller 파일 관리 API

POST	/api/files/create	
GET	/api/files/{id}/download	파일 다운로드

05 프로젝트 수행 방법

엔티티 설계

employees (직원)

속성명	한글명	설명	제약조건
id	식별자	고유 ID	필수, 고유, 자동 생성
employee_number	직원 번호	사원 번호	필수, 고유, 자동 부여, 수정 불가
employee_name	직원 이름	직원 이름	필수
email	이메일	직원 이메일	필수, 중복 금지
department_id	부서 ID	departments 테이블의 PK	필수,FK
position	직함	직원 직함	
hire_date	고용 날짜	입사일	필수
employee_state	직원 상태	직원의 재직 상태	기본값 `ACTIVE`
profile_image_id	프로필 이미지 ID	file_metadata 테이블의 PK	FK

departments (부서)

속성명	한글명	설명	제약조건
id	식별자	고유 ID	필수, 고유, 자동 생성
department_name	이름	부서 이름	필수, 고유
department_description	설명	부서 설명	필수
department_established_date	설립일	부서 설립일	필수

file_metadata (파일 메타정보 테이블)

속성명	한글명	설명	제약조건
id (PK)	식별자	고유 ID	필수, 고유 자동 생성
file_name	파일 이름	이미지 파일 이름	필수
content_type	파일 타입	이미지 MIME 타입 (jpg, jpeg, png, gif), csv, .log타입	필수, 이미지 타입, csv, .log파일만 허용
file_size	파일 크기	이미지 파일 크기	필수

change_logs (직원 정보 수정 이력 관리)

속성명	한글명	설명	제약조건
id	식별자	고유 ID	필수, 고유, 자동 생성
employee_id	대상 직원 ID	변경 이력이 발생한 직원의 ID (employees 테이블의 id와 외래키 연결)	필수, 외래키, 참조: employees(id)
type	이력 유형	이력 변경 유형 (CREATED, UPDATED, DELETED)	필수, Enum
memo	메모	사용자가 입력한 메모 내용	VARCHAR(255)
ip_address	IP 주소	요청을 발생시킨 클라이언트의 IP 주소	필수, IPv4 주소 저장, VARCHAR(20)
created_at	이력 생성 일시	변경 이력이 생성된 시간	필수, 자동 생성
details	변경 상세 내용	변경된 속성명, 이전 값, 이후 값 목록 ex: [{"propertyName": "직함", "before": "사원", "after": "대리"}]	JSONB 형식

change_log_details (직원 정보 수정 이력 상세 내용 관리)

속성명	한글명	설명	제약조건
id	식별자	고유 ID	필수, 고유, 자동 생성
change_log_id	변경 이력 ID	변경 이력의 ID (change_logs 테이블의 id와 외래키 연결)	필수, 외래키, 참조: change_logs(id)
property_name	변경 항목	변경된 속성	필수, VARCHAR(50)
before_value	변경 전 값	변경 전 속성 값	VARCHAR(255)
after_value	변경 후 값	변경 후 속성 값	VARCHAR(255)

backup_histories (백업 이력 테이블)

속성명	한글명	설명	제약조건
id (PK)	식별자	고유 ID	필수, 고유 자동 생성
operator	작업자	백업 작업자의 IP	필수
started_at	시작 시간	백업 시작 시간	필수
ended_at	종료 시간	백업 종료 시간	-
status	상태	백업 상태 (진행중, 완료, 실패, 건너뜀)	필수
backup_file_id (FK)	백업 파일	백업 파일의 CSV 파일 혹은 에러 로그의 .log 파일 메타 정보의 참조 ID	외래키

05 프로젝트 수행 방법

엔티티 설계

position_stats (직무별 직원 분포 통계 테이블)

속성명	한글명	설명	제약조건
id	식별자	고유 ID	필수, 고유 자동 생성
stat_date	해당 일자	집계를 며칠에 했는지	필수, 복합 Unique(stat_date, position_name, employee_state)
employee_count	직원 수	집계 일 기준 직무 인원 수	필수
created_at	생성시간	집계 데이터 생성 시간	필수
joined_employee_count	해당 일자 직무 입사자 수	해당 일에 몇 명이 해당 직무로 입사했는지	필수
left_employee_count	해당 일자 직무 퇴사자 수	해당 일에 해당 직무 인원이 얼마나 퇴사했는지	필수
position_name	직무 이름	통계 대상 직무의 이름	필수, 복합 Unique(stat_date, position_name, employee_state)
employee_state	직원 상태	직원 상태 별로 서로 다른 통계를 내기 위함	필수, 복합 Unique(stat_date, position_name, employee_state)

department_stats (부서별 직원 분포 통계 테이블)

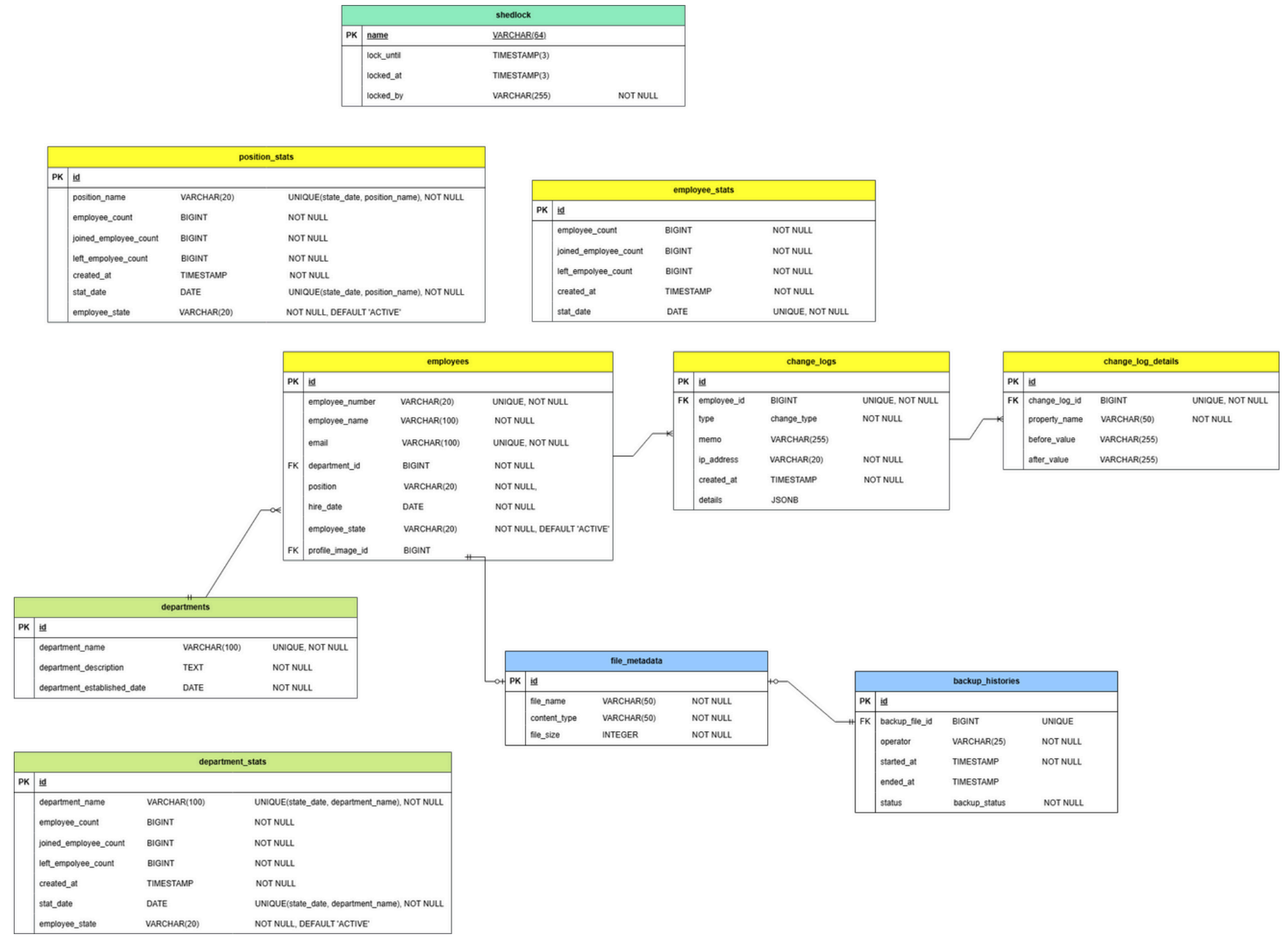
속성명	한글명	설명	제약조건
id	식별자	고유 ID	필수, 고유 자동 생성
stat_date	해당 일자	집계를 며칠에 했는지	필수, 복합 Unique(stat_date, department_id, employee_state)
employee_count	직원 수	집계 일 기준 부서 인원 수	필수
created_at	생성시간	집계 데이터 생성 시간	필수
joined_employee_count	해당 일자 부서 입사자 수	해당 일에 몇 명이 부서에 입사했는지	필수
left_employee_count	해당 일자 부서 퇴사자 수	해당 일에 해당 부서에서 몇 명이 퇴사했는지	필수
department_name	부서 이름	통계 대상 부서의 이름	필수, 복합 Unique(stat_date, department_name, employee_state)
employee_state	직원 상태	직원 상태 별로 서로 다른 통계를 내기 위함	필수, 복합 Unique(stat_date, department_name, employee_state)

employee_stats (일별 직원수 집계 테이블 / 전체 추이용)

속성명s	한글명	설명	제약조건
id	식별자	고유 ID	필수, 고유 자동 생성
stat_date	해당 일자	집계를 몇 일에 했는지	필수, 고유
employee_count	직원 수	집계 일 기준 직원 수	필수
created_at	생성시간	집계 데이터 생성 시간	필수
joined_employee_count	해당 일 입사자 수	해당 일에 몇 명이 입사했는지	필수
left_employee_count	해당 일 퇴사자 수	해당 일에 몇 명이 퇴사했는지	필수

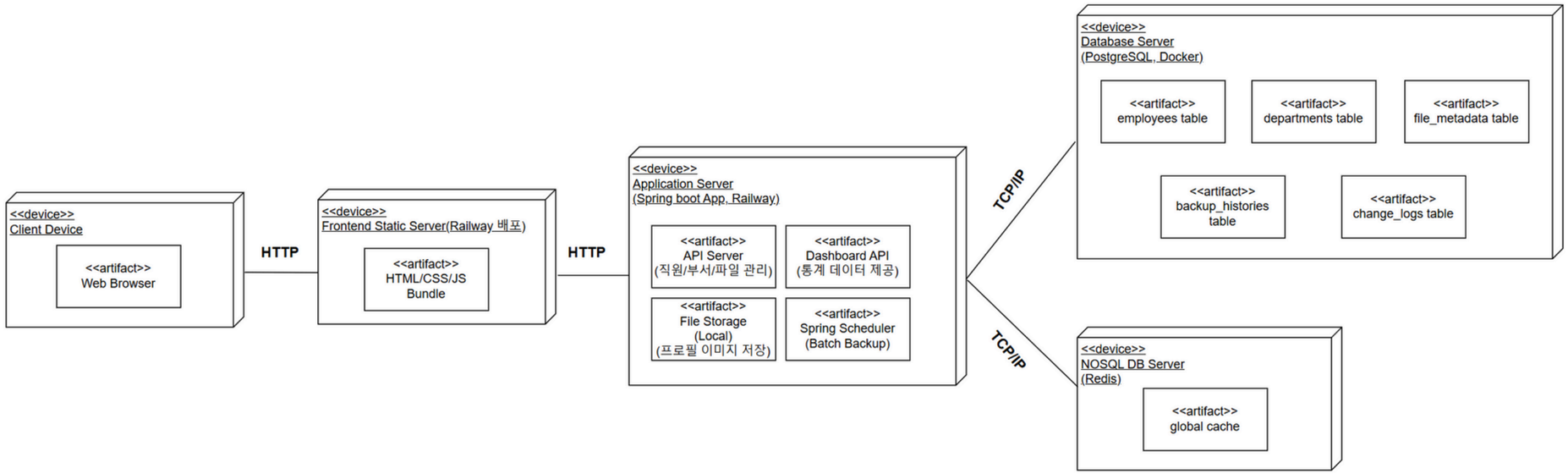
05 프로젝트 수행 방법

ERD



05 프로젝트 수행 방법

배포 다이어그램



https://drive.google.com/file/d/1f9AHrR6hVJf92zGToTTSVbU4GPyiWa3b/view?usp=sharing

06 트러블 슈팅

MapStruct 필드 매칭 오류 (직원 관리)

1. **발생한 문제**: DTO에 없는 필드 매핑 시 오류 발생

```
error: Several possible source properties for target property "id".
```

- EmployeeCreateRequest에는 없는 id 필드를 Employee로 매핑하려다 오류 발생.

2. 원인 분석: DTO에 없는 필드 매핑

- DTO(EmployeeCreateRequest)에는 id가 없지만,

```
public record EmployeeCreateRequest(
        String name,  1 usage
        String email,  2 usages
        Long departmentId,  1 usage
        String position,  1 usage
        LocalDate hireDate,  2 usages
        String memo  1 usage
) {
}
```

- Entity(Employee)에는 id가 존재 `private Long id;`
- MapStruct는 기본적으로 DTO 필드만 매핑하려 하기 때문에, 매핑 대상을 찾지 못해 오류 발생

06 트러블 슈팅

3. 해결 방법

```
@Mapping(target = "id", ignore = true)
```

를 추가해서 id 필드 매핑을 무시

4. 배운 점 및 개선 사항

- Entity에만 존재하고 DTO에는 없는 필드는 반드시 @Mapping(target = "xxx", ignore = true)를 명시해 매핑 누락 오류를 방지해야 함
- 여러 객체에서 동일한 이름의 필드가 있을 경우, MapStruct가 어떤 소스를 선택해야 할지 혼란스러워 할 수 있으므로, 명확하게 매핑을 무시하거나(ignore) 소스를 지정(source, expression)하여 오류를 예방해야 함

06 트러블 슈팅

PostgreSQL 한글 정렬 오류 개선 (부서 관리)

개발팀

기획팀

비서실

관리부서

마케팅팀

고객성공팀

1. **발생한 문제**: 한글 데이터를 정렬할 때 가나다 순으로 정렬은 되지만 글자수에 따라 정렬되는 문제가 있어서 기대하는 순서대로 정렬이 되지 않음

2. **원인 분석**: PostgreSQL의 기본 Collation 설정이 영어 기준으로 되어 있어, 한글 가나다 순의 정확한 정렬을 지원하지 않음
   → 찾아본 결과 애플리케이션 레벨(Spring/QueryDSL)에서는 별도로 정렬 방식을 제어할 수 없어, 데이터베이스 문제임을 확인

06 트러블 슈팅

3. 해결 방법: 부서명 컬럼의 Collation을 한국어 기준으로 변경했음
→ PostgreSQL에서 ko-x-icu Collation을 설정하여 한글 가나다 순서대로 정상 정렬 수행

```
ALTER TABLE departments
ALTER COLUMN department_name TYPE VARCHAR(100) COLLATE "ko-x-icu";
```

4. 배운 점 및 개선 사항: 정렬 문제는 단순 쿼리나 로직 수정으로 해결되지 않는 경우가 있는 것을 알았고, 데이터베이스 레벨에서 Collation 설정을 정확히 해야 언어 특성에 맞는 정렬 결과를 얻을 수 있다는 점을 배웠음

06 트러블 슈팅

복합 커서를 활용한 페이지네이션 로직 개선 (수정 이력 관리)

1, 발생한 문제

- cursor(마지막 조회 데이터의 정렬 기준 값)와 idAfter(마지막 조회 데이터의 id)를 서로 별개 커서로 인식하여 필터링 조건을 생성함

  → **페이지네이션 처리 시 데이터 중복 및 누락 문제 발생함**

2. 원인 분석

- cursor 값이 동일한 데이터가 여러 개 존재할 수 있음 (특히 ipAddress 값을 cursor로 사용하는 경우)

  → **cursor 값만 비교할 경우, 데이터들 간 정확한 경계를 설정할 수 없음**

06 트러블 슈팅

```
if ("desc".equalsIgnoreCase(sortDirection)) {
  cursorCondition.and(
      changeLog.createdAt.lt(cursorAt)
          .or(changeLog.createdAt.eq(cursorAt)
              .and(changeLog.id.lt(idAfter)))
```

```
if ("desc".equalsIgnoreCase(sortDirection)) {
  cursorCondition.and(
      changeLog.ipAddress.lt(cursorIp)
          .or(changeLog.ipAddress.eq(cursorIp)
              .and(changeLog.id.lt(idAfter)))
```

3. 해결 방법

– cursor와 idAfter를 조합하여 복합 커서로 활용

: WHERE (정렬 기준 〉 :cursor) OR (정렬 기준 = :cursor AND id 〉 :idAfter)

4. 배운 점 및 개선 사항

– 복합 커서 페이지네이션 개념 이해: cursor 값이 동일하여 각 데이터의 유일성이 보장되지 못하는 경우, 복합 커서를 활용하여 안정적인 페이지네이션 구현 가능

– 요구사항 및 API 명세 해석의 중요성: 단순한 스펙 이해를 넘어, 실질적인 데이터 구조와 예외 상황 등을 고려하는 습관이 필요함

06 트러블 슈팅

참조 무결성을 지킨 삭제 절차 설계의 중요성 (파일 관리)

1. **발생한 문제 – 파일 삭제 실패: FK 제약 위반 문제**
- 파일 메타데이터(file_metadata)를 삭제하려고 했으나, employees 테이블의 profile_image_id가 해당 파일을 참조하고 있어, 삭제 실패 “SQL FK constraint violation”가 발생했다.

2. **원인 분석**
- employees.profile_image_id는 file_metadata.id를 외래키(FK)로 참조하고 있음. 하지만 FK에 ON DELETE SET NULL이나 ON DELETE CASCADE 옵션이 없었기 때문에, 참조되고 있는 파일 메타데이터는 직접 삭제할 수 없었다.
- 즉, 단순히 파일만 삭제하면 된다고 생각했지만, DB 참조 무결성 제약 “Referential Integrity Constraint” 때문에 삭제가 막힌 상황이었다.

06 트러블 슈팅

3. **해결 방법**
   - 파일 삭제 요청이 들어오기 전에, 직원 테이블의 profile_image 키값이 null이 되어야지만, 파일 테이블 레코드와 실제 파일이 삭제될 수 있도록 함.
   - 직원 테이블에서 삭제 될 시 파일이 삭제되는 로직으로 변경함.

4. **배운 점 및 개선 사항**
   - 데이터베이스에서 외래키 제약이 설정된 경우, 참조 무결성을 깨지 않도록 삭제 절차를 신중하게 설계해야 한다는 것을 배웠다.
   - 단순한 파일 삭제도 실제로는 DB, 서비스 로직, 파일 시스템까지 연계된 작업임을 체감하고, 앞으로는 DDL 설계 단계부터 ON DELETE 정책을 명시하고, CRUD를 넘어 전체 데이터 흐름의 정합성을 기본으로 고려할 것이다.

06 트러블 슈팅

파일 작성시 OOM 이슈 대비 (데이터 백업)

1. **문제 사전 예방**
- CSV 파일에 데이터를 백업할 때 메모리 부족(OOM) 이슈가 발생할 수 있는 위험이 있다.

2. **원인 분석**
- 모든 데이터를 한꺼번에 메모리에 올린 뒤 저장(작성)하는 방식이면, 데이터량이 많을 때 메모리를 초과할 수 있다.

06 트러블 슈팅

3. **해결 방법**
- BufferedWriter를 사용해 20개씩 나눠 조회해 데이터를 파일에 스트리밍 방식으로 작성하고, 마지막에 flush를 통해 메모리 버퍼를 비워주는 방식으로 OOM을 예방했다.

4. **배운 점 및 개선 사항**
- 대용량 데이터 처리 시, 메모리에 전부 올리지 않고 스트리밍 방식으로 분할 처리해야 안정성과 성능을 모두 지킬 수 있다는 것을 배웠다.

06 트러블 슈팅

직원 수 변동 추이 데이터 조회 시 성능 저하 (대시보드)

```
List<EmployeeStats> employeeStatsList = statDateList.stream() Stream<LocalDate>
    .map(this::findEmployeeStatsByStatDate) Stream<EmployeeStats>
    .toList();
```

현재 statDateList의 size만큼 findEmployeeStats 쿼리를 날림
→ **N개의 쿼리 발생**

쿼리 최적화

```
employeeStatsRepository.findAllByStatDateIn(statDateList)
```

```
Hibernate:
    /* <criteria> */ select
        es1_0.id,
        es1_0.created_at,
        es1_0.employee_count,
        es1_0.joined_employee_count,
        es1_0.left_employee_count,
        es1_0.stat_date
    from
        employee_stats es1_0
    where
        es1_0.stat_date in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

IN절을 이용해 N → **1번의 쿼리로 개선**
(size가 500이라면 500번 → 1번)

인덱싱

```
Seq Scan on employee_stats  (cost=0.00..30.00 rows=8 width=44) (a
  Filter: (stat_date = ANY ('{2023-04-01,2023-04-02,2023-04-03,20
  Rows Removed by Filter: 992
Planning Time: 0.044 ms
Execution Time: 0.131 ms
```

```
Index Scan using idx_employee_stats_stat_date on employee_
  Index Cond: (stat_date = ANY ('{2023-04-01,2023-04-02,20
Planning Time: 0.048 ms
Execution Time: 0.026 ms
```

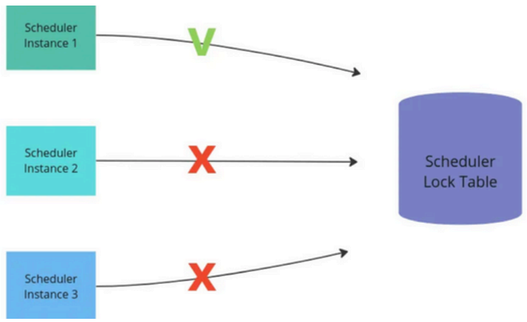
stat_date **인덱스**를 통해 약 0.1ms의 **성능 향상**

06 트러블 슈팅

스케쥴러가 여러 인스턴스/쓰레드에서 동시에 요청된다면? (대시보드)

```
@Override  no usages  GogiDosirak *
@Transactional
public void createTodayStats() {
  LocalDate currentDate = LocalDate.now(ZoneId.of( zoneId: "Asia/Seoul"));
  LocalDate prevDate = currentDate.minusDays( daysToSubtract: 1);
  long currentCount = employeeQueryRepository.count();
  if (employeeStatsRepository.existsByStatDate(currentDate)) {
    log.error("직원 통계 생성 실패: {}", currentDate);
    throw new RestException(ErrorCode.DUPLICATE_EMPLOYEESTATS);
  }
```

exists로 오늘 통계 데이터가 생성됐는지 확인하고 생성
→ **동시에** 요청이 들어올 경우 exists에서 모두 false를 반환
→ 동일한 날짜에 대한 통계가 **중복 생성될 위험**



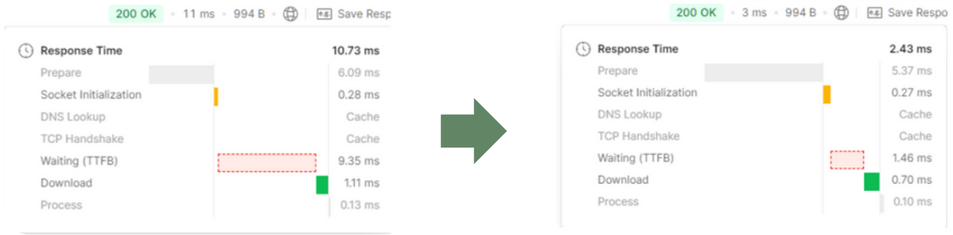
ShedLock 테이블을 이용해 멀티 인스턴스/쓰레드 **동시성 제어**

06 트러블 슈팅

ShedLock 사용 시, 로컬 캐시의 한계 (대시보드)

```
@CacheEvict(
    value = "positionDistribution",
    allEntries = true,
    cacheManager = "redisCacheManager"
)
public void createTodayStats() {
```

ShedLock으로 **한 인스턴스에서만** 데이터 생성
@CacheEvict로 **해당 인스턴스에서만** 캐시를 삭제 → 갱신
다른 인스턴스는 **무효화되지 않은** 이전 캐시를 계속 사용
→ 최신 DB 데이터와 **불일치 발생**



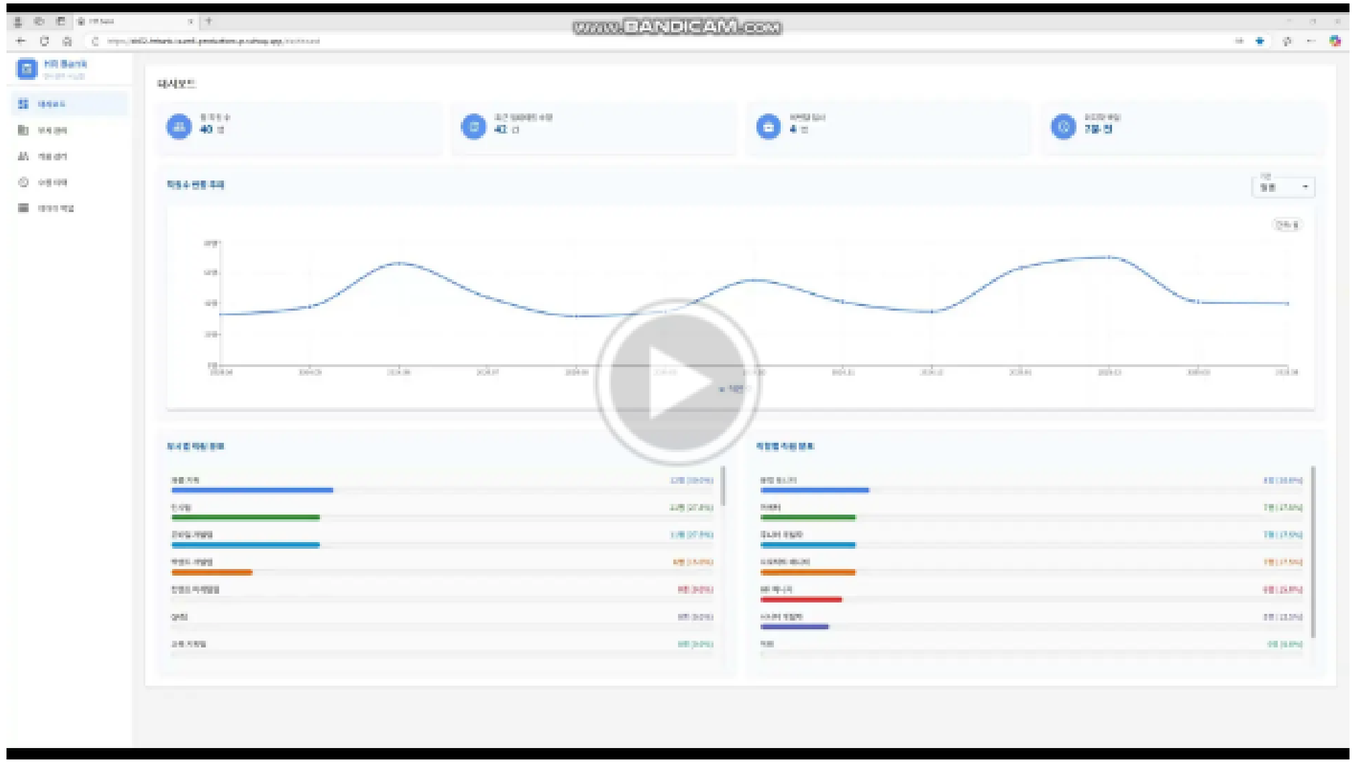
cf) 로컬캐시 사용 시, ~10ms 만큼의 **성능 향상** (데이터 500개)

```
127.0.0.1:6379> keys *
1) "departmentDistribution::ACTIVE2025-04-28"
2) "employeeTrend::day"
3) "employeeTrend::quarter"
4) "positionDistribution::ACTIVE2025-04-28"
5) "employeeTrend::week"
6) "employeeTrend::month"
7) "employeeTrend::year"
```

글로벌 캐시 사용 → 모든 인스턴스가 **동일한** 캐시 저장소를 참조

07 프로젝트 수행 결과

시연 영상



07 프로젝트 수행 결과

배포 링크

https://sb02-hrbank-team6-production.up.railway.app/dashboard

08 팀 자체 평가

아쉬운 점

- 대용량 데이터 처리 경험 및 성능 최적화 경험을 기대했으나, 실제로는 데이터 규모가 작아 충분한 경험을 쌓기 어려웠음

잘한 점

- 요구사항을 충실히 반영한 설계를 기반으로 정확하게 API를 구현
- 기능별 역할 분담, 기능 단위의 브랜치 전략을 통해 충돌을 최소화하고 협업 효율을 높임
- 에러 발생 시 팀원 간 빠른 소통과 협력을 통해 문제를 신속하게 해결

느낀 점

- 기능별로 역할을 나눠 작업했지만, 하나의 시스템을 완성하기 위해서는 유기적인 협력이 필요했으며, 세세한 부분까지 공유하며 즉각적으로 소통하는 것이 중요함을 실감했음

09 질문과 답변

감사합니다